Problem 2

Problem Description

In this problem you will train a neural network to classify points with features $x_0$ and $x_1$ belonging to one of three classes, indicated by the label $y$. The structure of your neural network is up to you, but you must describe the structure of your network, training parameters, and report an accuracy for your fitted model on the provided data.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- Visualization of provided data
- Visualization of trained model with provided data
- Trained model accuracy
- Discussion of model structure and training parameters

Imports and Utility Functions:

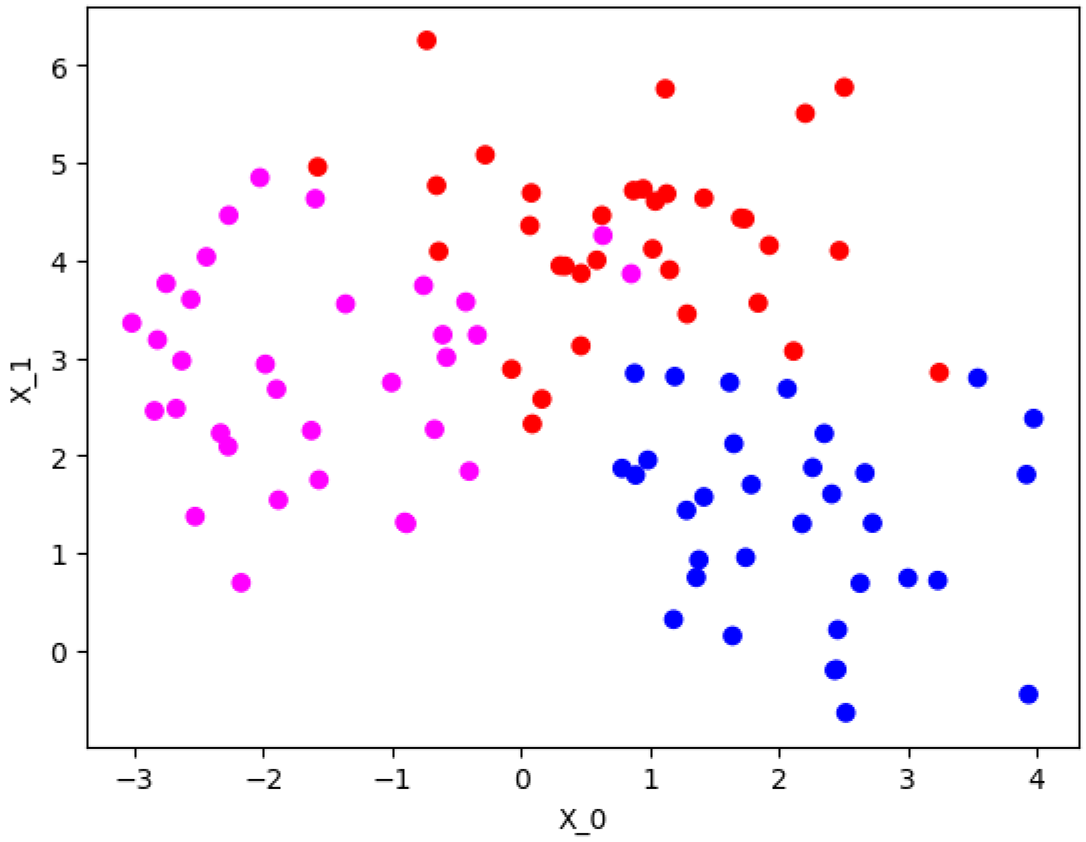
In [7]:
```
import torch
import torch.nn as nn
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import torch.nn.functional as F
from torch import optim


def dataGen():
    # random_state = 0 set so generated samples are identical
    x, y = datasets.make_blobs(n_samples = 100, n_features = 2, centers = 3, random_state = 0)
    return x, y
```

```
def visualizeModel(model):
    # Get data
    x, y = dataGen()
    # Number of data points in meshgrid
    n = 100
    # Set up evaluation grid
    x0 = torch.linspace(min(x[:,0]), max(x[:,0]),n)
    x1 = torch.linspace(min(x[:,1]), max(x[:,1]),n)
    X0, X1 = torch.meshgrid(x0, x1, indexing = 'ij')
    Xgrid = torch.vstack((X0.flatten(),X1.flatten())).T
    Ypred = torch.argmax(model(Xgrid), dim = 1)
    # Plot data
    plt.scatter(x[:,0], x[:,1], c = y, cmap = ListedColormap(['red','blue','magenta']))
    # Plot model
    plt.contourf(Xgrid[:,0].reshape(n,n), Xgrid[:,1].reshape(n,n), Ypred.reshape(n,n), cmap = ListedColormap(['red', 'b
    plt.xlabel('$x_0$')
    plt.ylabel('$x_1$')
    plt.show()
```

Generate and visualize the data

Use the `dataGen()` function to generate the x and y data, then visualize with a 2D scatter plot, coloring points according to their labels.

In [8]:
```
# YOUR CODE GOES HERE
x,y = dataGen()
plt.scatter(x[:,0], x[:,1],c=y,cmap= ListedColormap(['red','blue','magenta']))
plt.xlabel('X_0')
plt.ylabel('X_1')
plt.show()
```



Create and train a neural network using PyTorch

Choice of structure and training parameters are entirely up to you, however you will need to provide reasoning for your choices. An accuracy of 0.9 or more is reasonable.

Hint: think about the number out nodes in your output layer and choice of output layer activation function for this multi-class classification problem.

In [9]:
```
# YOUR CODE GOES HERE


x = torch.Tensor(x)
y = torch.Tensor(y)
```

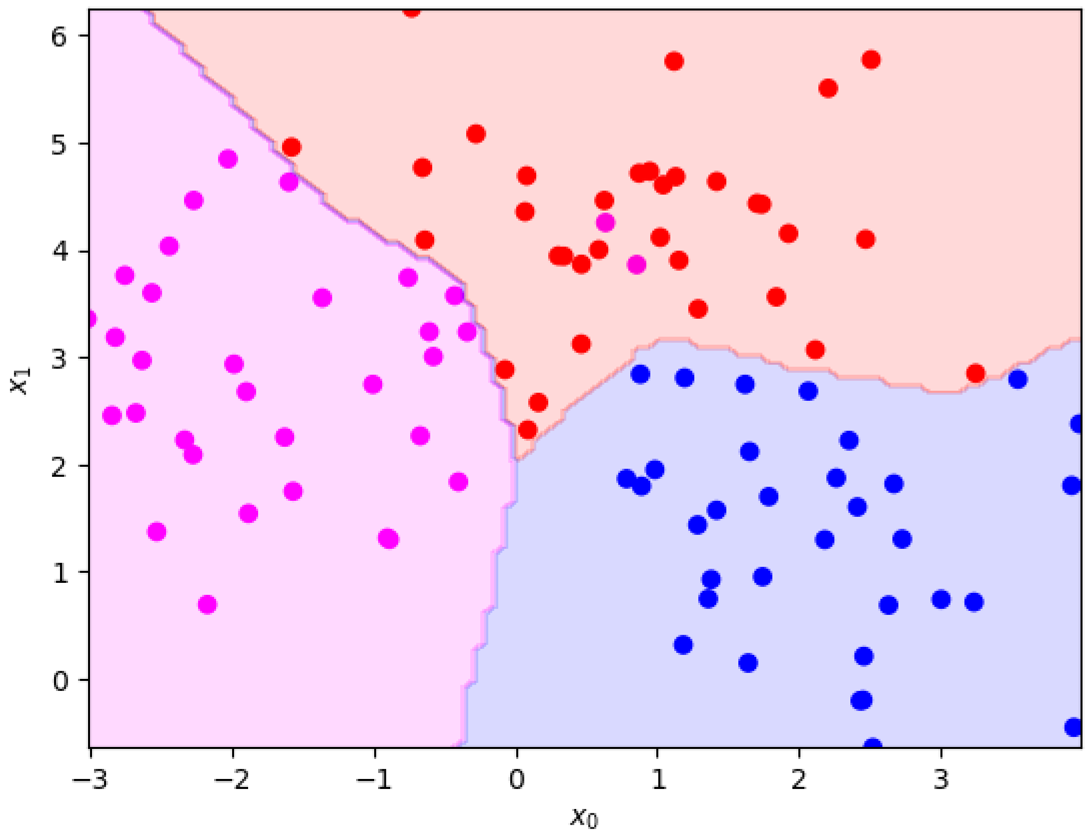
```
class Net_2_layer(nn.Module):
    def __init__(self, N_hidden=16, N_in=1, N_out=3, activation = F.relu):
        super().__init__()

        self.lin1 = nn.Linear(N_in, N_hidden)
        self.lin2 = nn.Linear(N_hidden, N_hidden)
        self.lin3 = nn.Linear(N_hidden, N_hidden)
        self.lin4 = nn.Linear(N_hidden, N_out)
        self.act = activation


    def forward(self,x):
        x = self.lin1(x)
        x = self.act(x)
        x = self.lin2(x)
        x = self.act(x)
        x = self.lin3(x)
        x = self.act(x)
        x = self.lin4(x)
        x = F.softmax(x, dim = 1)
        return x

model = Net_2_layer(N_hidden = 32, N_in = 2, N_out = 3, activation = F.relu)
loss_curve =[]

lr = 0.005
epochs = 1000
loss_fcn = nn.CrossEntropyLoss()


opt = optim.Adam(params = model.parameters(), lr=lr)

for epoch in range(epochs):
    out = model(x)
    loss = loss_fcn(out,y.long())
    loss_curve.append(loss.item())

    _, predicted = torch.max(out, 1)
    correct = (predicted == y).sum().item()
    accuracy = correct / len(y)

    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch} of {epochs}... \tAccuracy: {accuracy:.2f}")
    opt.zero_grad()
    loss.backward()
    opt.step()
```

```
Epoch 0 of 1000...      Accuracy: 0.32
Epoch 40 of 1000...     Accuracy: 0.90
Epoch 80 of 1000...     Accuracy: 0.93
Epoch 120 of 1000...    Accuracy: 0.96
Epoch 160 of 1000...    Accuracy: 0.97
Epoch 200 of 1000...    Accuracy: 0.97
Epoch 240 of 1000...    Accuracy: 0.98
Epoch 280 of 1000...    Accuracy: 0.98
Epoch 320 of 1000...    Accuracy: 0.98
Epoch 360 of 1000...    Accuracy: 0.98
Epoch 400 of 1000...    Accuracy: 0.98
Epoch 440 of 1000...    Accuracy: 0.98
Epoch 480 of 1000...    Accuracy: 0.98
Epoch 520 of 1000...    Accuracy: 0.98
Epoch 560 of 1000...    Accuracy: 0.98
Epoch 600 of 1000...    Accuracy: 0.98
Epoch 640 of 1000...    Accuracy: 0.98
Epoch 680 of 1000...    Accuracy: 0.98
Epoch 720 of 1000...    Accuracy: 0.98
Epoch 760 of 1000...    Accuracy: 0.98
Epoch 800 of 1000...    Accuracy: 0.98
Epoch 840 of 1000...    Accuracy: 0.98
Epoch 880 of 1000...    Accuracy: 0.98
Epoch 920 of 1000...    Accuracy: 0.98
Epoch 960 of 1000...    Accuracy: 0.98
```

Visualize your trained model

Use the provided `visualizeModel()` function by passing in your trained model to see your models predicted function compared to the provided data

In [10]:
```
# YOUR CODE GOES HERE
visualizeModel(model)
```



Discussion

Report the accuracy of your trained model on the generated data. Discuss the structure of your network, including the number and size of hidden layers, choice of activation function, loss function, optimizer, learning rate, number of training epochs.

YOUR ANSWER GOES HERE

The accuracy after training the model was found to be 0.98.

There was two input layer(N_in) and three output layer(N_out). As it is a classification problem with three classes so three output layers were used.

There are three hidden layer was used with 32 layers of preceptron. Smaller preceptron layers were tried i.e., 2,4 etc before using 32 but 32 gave us a good accuracy of 0.98. Also, the higher number of hidden layers and neurons increases the complexity which improves the overall trainng of the model. Hence, 3 hidden layers and 32 neurons provided a good fit which in turn gave us a good accuracy.

Along with that the activation function used was Relu. Because it leads to a fast convergence i.e., it is faster to compute and it also helps in overcoming vanishing gradients. Relu was used in all the layers for activation except for the last one. Along with Relu, the softmax activation function was used in the final layer.

The loss function used was nn.CrossEntropyLoss because it is a multi-class classification problem and not a regression problem.

The optimizer used was Adam as it converges faster than any other optimizer.

As a smaller learning rate leads to a better fit of the curve. So, the learning rate used was 0.005. A lower and higher learning rates were used too but this provided the results needed for the problem.

The number of iterations or number of training epochs was kept at 1000. This was the best to use. As smaller number of training epoch provided a higher MSE and a larger number of training epochs provided an overfitting curve.

```
In [ ]:
```